Naval Research Laboratory
Washington, DC 20375-5000

NRL Report 9124

Design Changes in the Software Cost-Reduction Project

L. J. CHMURA, A. F. NORCIO, AND T. J. WICINSKI

Human-Computer Interaction Laboratory
Information Technology Division

June 30, 1988

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 9124	5 MONITORING ORGANIZATION REPORT NUMBERS
6a NAME OF PERFORMING ORGANIZATION Naval Research Laboratory	6b OFFICE SYMBOL (If applicable) Code 5530
6c ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 61013N; PROJECT NO:; TASK NO: R1947; WORK UNIT ACCESSION NO:

11 TITLE (Include Security Classification)
Design Changes in the Software Cost-Reduction Project

12 PERSONAL AUTHOR(S)
Chmura, L. J., Norcio, A. F., and Wicinski, T. J.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM 1/80 TO 12/87	1988 June 30	28

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software engineering; Data analyses; Data collection; SCR project

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents analyses of early design changes recorded on change report forms (CRFs) from the Software Cost-Reduction (SCR) project at the Naval Research Laboratory. SCR engineers are redesigning the operational flight program for the Navy's A-7E aircraft by using software engineering principles such as information hiding. The two major goals are to demonstrate the effectiveness of these techniques in developing, for example, easy to change real-time software and to provide high-quality development goals for other to follow. The first part of this report describes the SCR development goals for others to follow. The first part of this report describes the SCR development effort and examines how well it is meeting its goals. Results are presented as time-based trends and are compared to similar data reported from other projects. The second part examines the time-based ratios between SCR change data and personnel activity and the possible general use of these ratios as indicators of design progress. Two similar ratios are identified that show promise as indicators of design incompleteness.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL L. J. Chmura	22b TELEPHONE (Include Area Code) (202) 767-3249
	22c OFFICE SYMBOL Code 5533

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

DESIGN CHANGES IN THE SOFTWARE COST-REDUCTION PROJECT

INTRODUCTION

This report presents analyses of early design changes proposed and made by software development engineers working on the Software Cost-Reduction (SCR) project at the Naval Research Laboratory (NRL). The remainder of this section is an overview of NRL's SCR project and Software Technology Evaluation (STE) project. The second section describes the techniques and strategies that were used in collecting and categorizing the data. The third section is a detailed discussion of the change and error data. The final two sections contain the analyses of the data and their implications.

Software Cost-Reduction Project

Since 1978 the Naval Research Laboratory, in cooperation with the Naval Weapons Center, has been redeveloping version 2 of the operational flight program for the A-7E aircraft [1]. Software engineering techniques such as formal requirements specification [2], information hiding [3], abstract interfaces [4], and cooperating sequential processes [5] are being used. This research effort is referred to as the Software Cost-Reduction (SCR) project.

The goals of the project are to demonstrate the feasability of using selected software engineering techniques in developing complex, real-time software and to provide a model for software design. The claimed advantage of the selected software engineering techniques is that they facilitate the development of software that is easy to change and maintain. Reference 1 provides a thorough overview of the SCR project and Ref. 6 is a complete discussion of the project's software requirements. Reference 7 provides a detailed description of the module design structure. Figure 1 is an example of a module interface specification (i.e., a design specification) taken from a recent version of the specification for the device interface module [8]. Reference 9 describes a standard organization for such specifications.

The SCR project terminated at the end of 1987 after implementing three subsets of the operational flight program requirements. The subsets are being evaluated and tested by using ground-based test facilities.

SCR project data have been collected in three areas: personnel activity [10], changes to requirements [11], and changes to design and code. This report is the first published analysis of SCR design and code change data.

Software Technology Evaluation Project

The STE project is a separate research project from the SCR project in terms of goals, staffing, and funding.* The goal of the STE project is to evaluate alternative software development technologies. A major task of the STE project, therefore, is to provide the basis for an objective evaluation of the methodology used in the SCR project.

*The project was at one time funded by the DoD STARS Program as Measurement Area Task G-06
Manuscript approved February 2, 1988.

DI.VI: VISUAL INDICATORS (Auto-Cal and Non-Align Indicators)

1. Introduction

There are two visual indicators controlled by the OFP on the A-7 aircraft, one that can, and one that cannot, be seen by the pilot during flight. These are currently labeled "IMS Non-aligned" and "Auto-Cal", respectively. Each can be on steady, on blinking, or off.

2. Interface overview

2.1. ACCESS PROGRAM TABLE

Program	*Parameters*	*Description*	*Undesired events*
+G S_AUTOCAL_INDICATOR+	p1: VIS_ind_cntrl; O: I	'+Auto-cal+'	None
+S_AUTOCAL_BLINK_RATE+	p1: real; I	blink rate	
+G S_NON_ALIGN_INDICATOR+	p1: VIS_ind_cntrl; O: I	'+Non-align+'	
+S_NON_ALIGN_BLINK_RATE+	p1: timeint; I	"rate"	

Effects

+S_AUTOCAL_INDICATOR+	IF p1=On THEN "Auto-Cal" indicator turned on; IF p1=Off THEN "Auto-Cal" indicator turned off; IF p1=$Intermittent$ THEN "Auto-Cal" indicator turned on and off at the rate set by +S_AUTOCAL_BLINK_RATE+, or at the system default rate.
+S_AUTOCAL_BLINK_RATE+	When commanded to blink, the blink "rate" will be p1.
+S_NON_ALIGN_INDICATOR+	IF p1=On THEN "Non Align" indicator turned on; IF p1=Off THEN "Non Align" indicator turned off; IF p1=$Intermittent$ THEN "Non Align" indicator turned on and off at the rate set by +S_NON_ALIGN_BLINK_RATE+, or at the system default rate.
+S_NON_ALIGN_BLINK_RATE+	When commanded to blink, the blink "rate" will be p1.

3. Local type definitions

VIS_ind_cntrl — Enumerated: On, Off, $Intermittent$

4. Dictionary

'+Auto-cal+' — The state of the auto-cal indicator as last set by +S_AUTOCAL_INDICATOR+.

'+Non-align+' — The state of the non-align indicator as last set by +S_NON_ALIGN_INDICATOR+.

5. Undesired event dictionary

None

6. System generation parameters

#Autocal blink default#* — Type: timeint. Default blink interval for "Auto-Cal" indicator.

#Auto-Cal init state#* — Type: VIS_ind_cntrl. The system-load-time value for '+Auto-cal+'.

#Nonalign blink default#* — Type: timeint. Default blink interval for "Non-Align" indicator.

#Non-Align init state#* — Type: VIS_ind_cntrl. The system-load-time value for '+Non-align+'.

* The value of this system generation parameter may be set by user software. See section 2.2 of the introduction to this document.

Fig. 1 -- Example of module interface specification

The approach followed in the STE project is to monitor, evaluate, and compare software development technologies used in different software projects. The monitoring and evaluating processes consist of goal-directed data collection and analyses techniques [12].

COLLECTION OF CHANGE DATA

From 1980 until early 1985, SCR project engineers reported design and code problems, and suggested design changes. They logged their modification activity to baselined (i.e., published and change-controlled) interface specifications, pseudo code, and TC-2 code† on Change Report Forms (CRFs). Figure 2 is an example of a completed CRF. There are two reasons for this procedure. First, it is required by SCR project configuration management (CM) procedures. Second, such data are needed by STE researchers for evaluating achievement of SCR project goals. The specific design of the CRF form is based on a goal-directed data collection approach [13]. In 1985, the use of paper CRFs was discontinued. Since then, SCR engineers have noted problems and proposed changes by using a computer-based configuration management tool.

STE researchers have validated primarily those CRFs that have been resolved either by official acceptance and incorporation into the baselined documentation or by official rejection of the proposed change. Ideally, validation should be a continuing activity that occurs as CRFs are generated and resolved. Validation of SCR CRFs, however, has tended to be an aperiodic activity in which large groups of CRFs are validated at one time. The validation consists of a checking completeness, accuracy, etc. It often includes discussions with persons who submitted the CRFs, authors of affected documents, and SCR CM personnel. A major validation point concerns what constitutes a design or code change. A proposed change must be stated by a simple declarative sentence; the change comprise alterations to one or more baselined interface specifications or implementation documents. Basically, the view taken is that a change is conceptual. In addition, a change must have a unique basis—error correction, adaption to outside change, improvement, or other (see Fig. 2). The basis for this scheme follows the scheme developed by Swanson [14]. Thus, a change that is described in one CRF similar to a change in a CRF resolved and implemented in earlier baselines (i.e., a change that requires completion or correction to earlier baselined alterations) is a unique or new change. A proposed change that is rejected obviously results in no alterations.

This definition of a design or code change can cause problems. Occasionally a CRF is submitted that incorporates more than one change, and different engineers sometimes submit the same change on different CRFs. For example, it is not unusual for a CRF to describe two conceptual changes as:

> "The last sentence of the description is ambiguous. Replace it with . . .
> Note also that the word *descriptor* is misspelled."

A workable solution used by STE researchers for dealing with these situations is to split submitted CRFs that incorporate more than one change into individual CRFs so that each CRF describes only one change. Multiple CRFs that describe identical changes are consolidated into a single CRF. One result of this policy is that a one-to-one correspondence does not exist between submitted CRFs and validated CRFs. The other result of course, is that a one-to-one correspondence does exist between proposed changes and validated CRFs.

Other sections of the CRF also cause difficulties. One difficulty is determining the basis of an accepted change. Another problem is that it is not sufficient to define an error as a discrepancy

†TC-2 code is the assembly language code for the IBM System 4 PI model TC-2 computer. The A-7E operational flight program runs on this machine.

SCR PROJECT: DESIGN AND CODE CHANGE REPORT FORM

CRF ID: 227

SUGGESTED CHANGE [Filled In By CRF Originator]

Originator: Glasgow Date: 21 April 93

Change Description (Identify all affected documents /versions and pages):

FC spec, ver 3. p. 32, see EC-102

Effort For Understanding And Specifying Change

0 — 1 work hour — 1 work day — 1 work week — 1 work month — ∞

What activity led to discovery of need for change?

- X Project Activity
  - 1 Design creating ECT, ECT2.TO
  - ____ Pseudo Code
  - ____ Code
  - ____ Module Test
  - ____ Subset Test
  - ____ Miscellaneous
- ____ Non-Project Activity

CHANGE CLASSIFICATION [Filled In By Originator And Change Engineers]

Basis For Change

- ____ Correction of original error
- ____ Correction or completion of earlier change, CRF ____
- ____ Adaptation to requirements CRF ____, that is, a
  - ____ requirements error
  - ____ expected requirements change
  - ____ unexpected requirements change
- X Adaptation to change in support environment
- ____ Improvement in
  - ____ performance
  - X clarity, or maintainability
- ____ Other

[see back side]

7 Feb 89

CHANGE CLASSIFICATION [Filled In By Originator And Change Engineers]

Change Areas: [Mark all updated by change]

- ____ actual input-output device, formats, or protocols
- ____ timing of system functions
- ____ set of processes or their timing
- ____ UE handling

(Number) Baselined Bottom-Level Design Modules Updated: (1) PC, 0

(Number) Baselined Bottom-Level Design Interfaces Updated: (1) PC, 10

(Number) Baselined Documents Updated: (1) PC spec.

ERROR CLASSIFICATION [Filled In By Originator And Change Engineers]

Error Cause(s)

- ____ Clerical
- ____ Designer or coder misunderstood
  - ____ Requirements
  - ____ Interface specification
  - ____ Pseudocode
  - ____ Pseudocode language
  - ____ Programming environment
  - ____ Uses hierarchy
  - ____ Other
- ____ Other

Techniques Leading To Error Discovery And Resolution: ____

RESOLUTION LOG [Filled In By Change Engineers]

Engineer: Sinsonte Date: 12 July 93

Effort For Understanding And Specifying Change

0 — 1 work hour — 1 work day — 1 work week — 1 work month — ∞

DISPOSITION [Filled In By Head Of Configuration Control]

- X Accepted as described above
- ____ Rejected because

Signature: Sinsonte Date: 12 July 93

Fig. 2 - Completed CRF form

between a specification and its implementation. For example, it is sometimes difficult to decide if a CRF describes an inadequate interface design (i.e., an error) or if it simply describes a better design (i.e., an improvement). The only reasonable solution to this problem has been to let SCR lead engineers decide between these situations. A second problem is determining whether or not a change is a correction or a completion of an earlier change that has already been baselined. The fact is, after a long period of time or after many versions of a document, authors frequently forget earlier changes that had addressed the same issues presented in current CRFs. For each of the CRFs reported in this study, STE researchers have reviewed all versions of all documents baselined prior to resolution of the CRF and discussed all questions with lead SCR engineers. This is a laborious process, but it is necessary to ensure that corrections or completion errors are properly identified.

Finally, the SCR project's CM procedures are not perfect. Validators have found CRFs that have not been resolved but, nevertheless, have been implemented in published specifications. The only reasonable solution for this is to resolve these CRFs with the date of the latest issued baseline specification and to submit additional CRFs for remaining aspects of the change. Validators have also found modifications for which there were no corresponding CRFs. The policy for this has been to submit CRFs and record them as immediately resolved with the date of issue of the baseline specification.

OVERVIEW OF EARLY SCR CHANGE DATA

General

This study reports on 325 validated CRFs that were resolved before 1 January 1984. During this period, engineers had submitted 424 CRFs. The 325 validated CRFs reported here map 296 (70%) of those submitted that were resolved by SCR CM personnel by this date. Figures 3 and 4 are profiles of resolution activity for these proposed changes. By January 1984, ~47,500 person hours had been expended on the SCR project. The 400 hours of resolution effort accounted for ~1% of project activity. Table 1 shows the distribution of the CRFs categorized by the originators' activities when the CRFs were generated. In addition, only 15% of SCR project hours were spent on pseudo coding, coding, and testing activities. Thus, the changes reviewed here can be characterized as changes that are typically proposed and made early in software development, which contrasts with changes reported elsewhere [15-17].

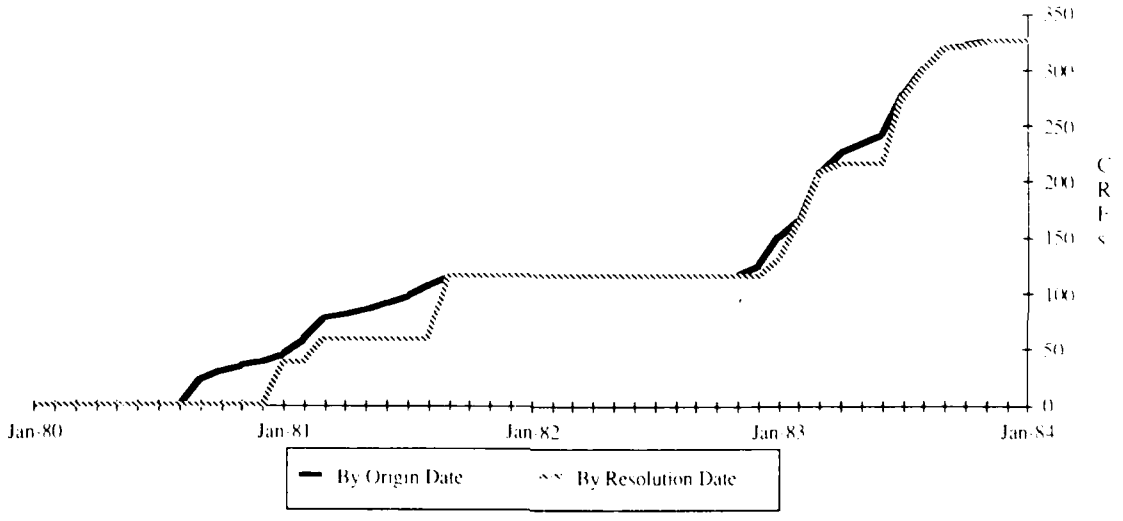

Fig. 3 — CRF accumulation

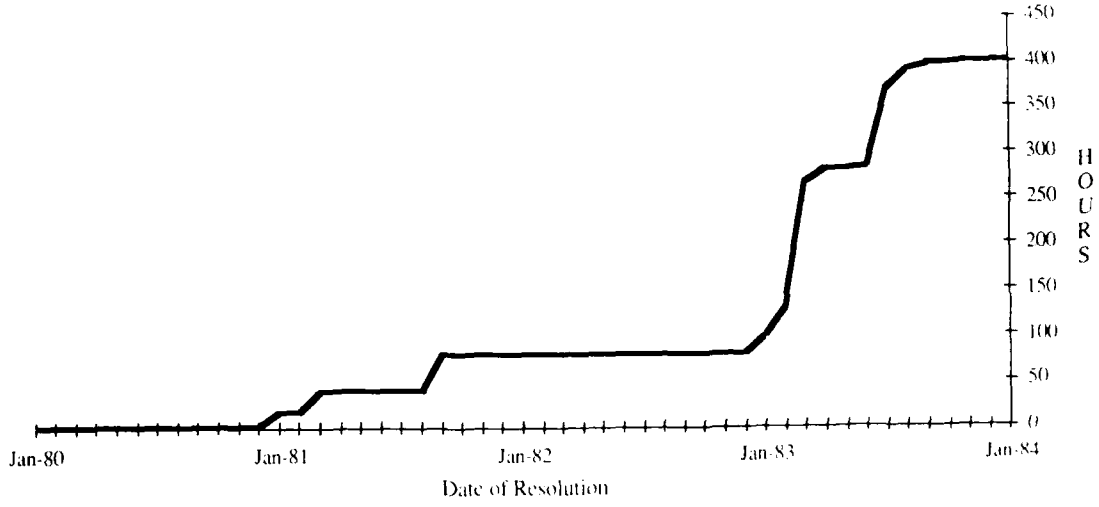

Fig. 4 — Cumulative effort in resolving CRFs

Table 1—Activities Leading to CRF Origination

Project Activity	Total CRFs	Percentage
Design (e.g., module interface specification)	209	6
Pseudo code	53	16
Code	1	0
Test	26	8
Miscellaneous	15	5
Unknown	5	?
Total	309	95
Nonproject activity (e.g., CRF validation)	16	5
Total	325	100

Twenty-eight (9%) of the 325 proposed changes were rejected; this required ~ 18 hours (4%) of the total hours expended on the changes (Figs. 5 and 6). The 9% figure is small compared to both the 37% figure reported by Day [18] for major maintenance updates to an operational Army command and control system and the 20% figure reported by Shooman and Bolsky [19] for errors discovered and corrected during test and integration of a modest-size control program at Bell Telephone Laboratories. The 4% effort figure is comparable to the 3% figure reported in Ref. 18. Care must be taken with these comparisons, however. These two figures are from different times in different project life cycles, and it is not clear if there is a common definition of change. More importantly, SCR requirements changes are a separate SCR CM concern and are not incorporated in the data reported here [11].

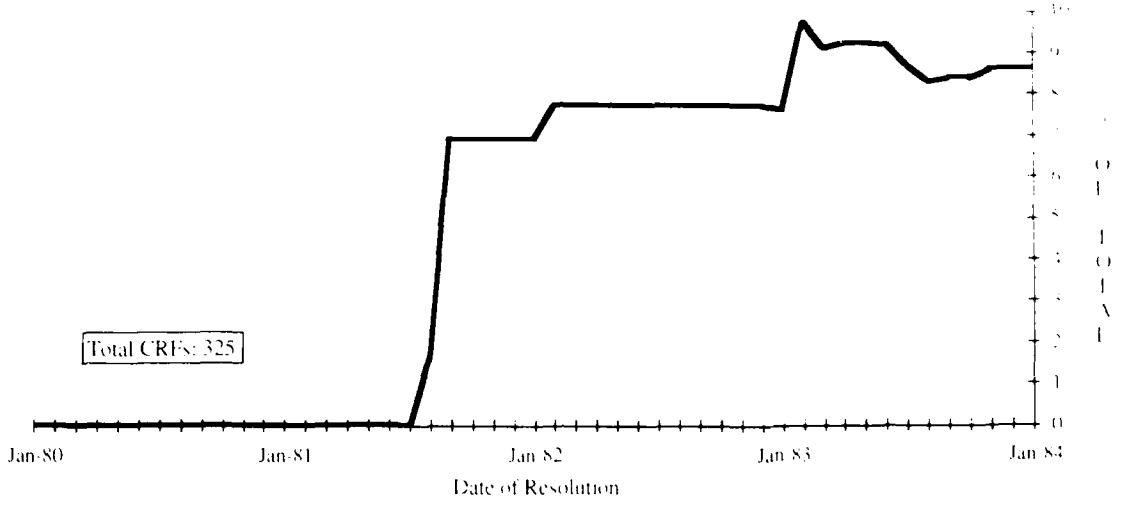

Fig. 5 — Rejected CRFs: percentage of total

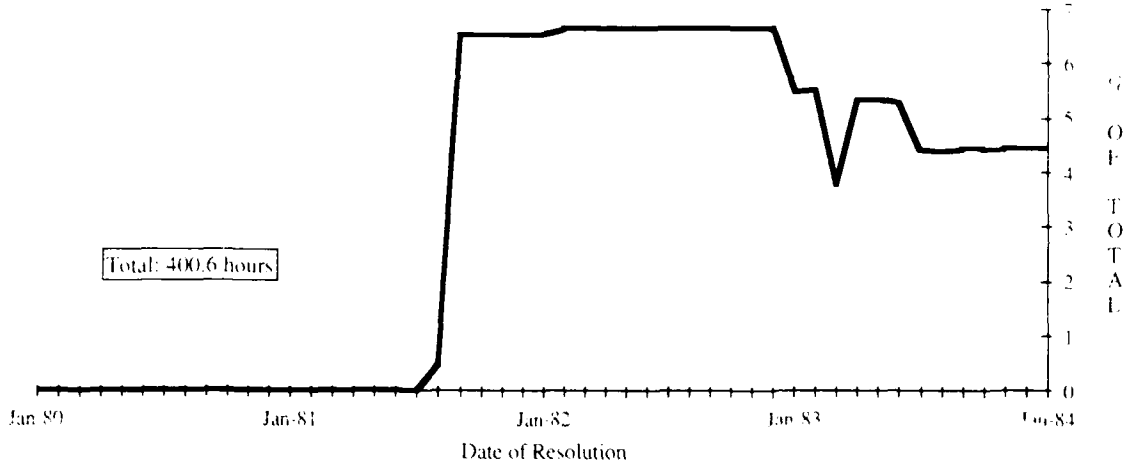

Fig. 6 — Rejected CRF resolution effort: percentage of total

The remaining 297 accepted CRFs resulted in modifications to baselined items. Table 2 shows the bases for these changes. None of the changes were the result of changes to the software requirements specification. This can probably be attributed to the following:

- an extensive requirements specification was generated prior to design [6],
- the requirements specification has been shown to be relatively error free and remarkably free of ambiguities [11],
- as noted earlier, the changes reported are *early* changes, and
- the SCR project is redeveloping software for a fixed operational version of the A-7E flight software.

Table 2—Bases of Accepted CRFs

	Total CRFs	Percentage
Error Corrections		
Original	144	48
Continuation of completion	55	19
Total	199	67
Modifications		
Adaption to requirements change	0	0
Adaption to support environment change	0	0
Improvement in performance	2	1
Improvement in clarity	89	30
Other	7	2
Total	98	33

Actually, all 297 changes required updates to only 47 baselined module interface specifications, most of which are packaged in two documents. The primary reason for this is that no module implementation documents (which include pseudo code) were baselined before January 1984. In other words, the 297 changes can be considered to be early design changes.

The percentage of error corrections (see Table 2 and Fig. 7) is high compared to data reported for other development efforts [15,17,19], but this is decreasing. The proportion of total CRF effort spent on error corrections (Fig. 8) contrasts sharply with the 17% figure reported by Lientz and Swanson [20] for commercial data processing software maintenance efforts and the 21% figure reported by Day [18]. This percentage of error correction effort is also decreasing. Note again however, that SCR requirements document change data are not included in this summary.

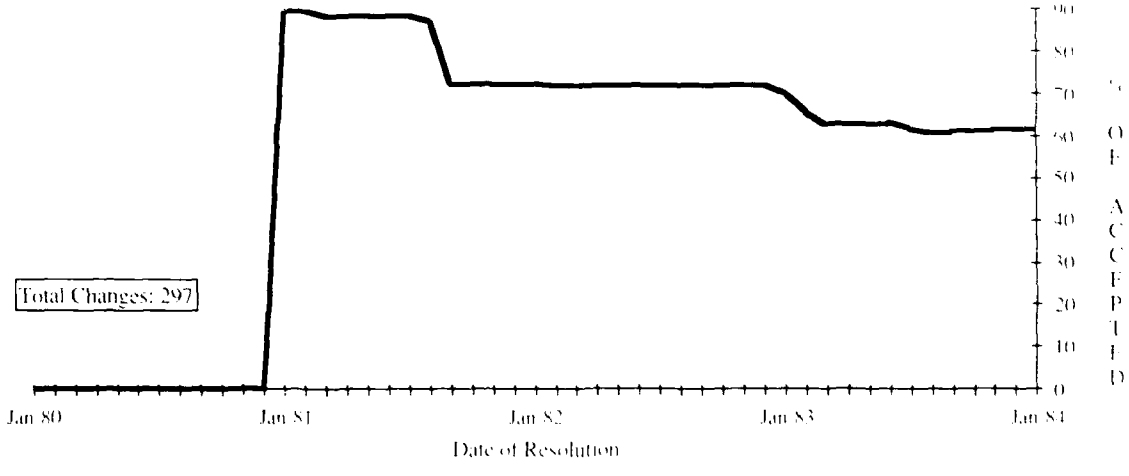

Fig. 7 — Error corrections: percentage of accepted changes

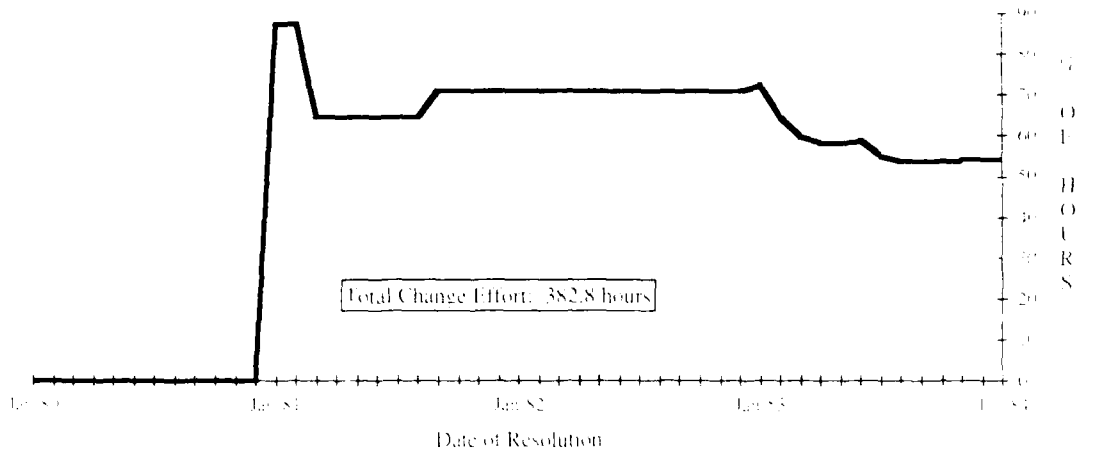

Fig. 8 — Error correction effort: percentage of accepted CRF resolution effort

The proportion of error corrections that involve completing or correcting a prior change (Fig. 9) is large compared to the 6% to 12% range of figures reported by others [15,17,21] and seems to be incrementally increasing. The 12% figure is computed from data given by Weiss [21] and Weiss and Basili [17]. This large proportion could be the result of the many hours spent by STE and SCR engineers in assuring the correct identification of correction and completion errors.

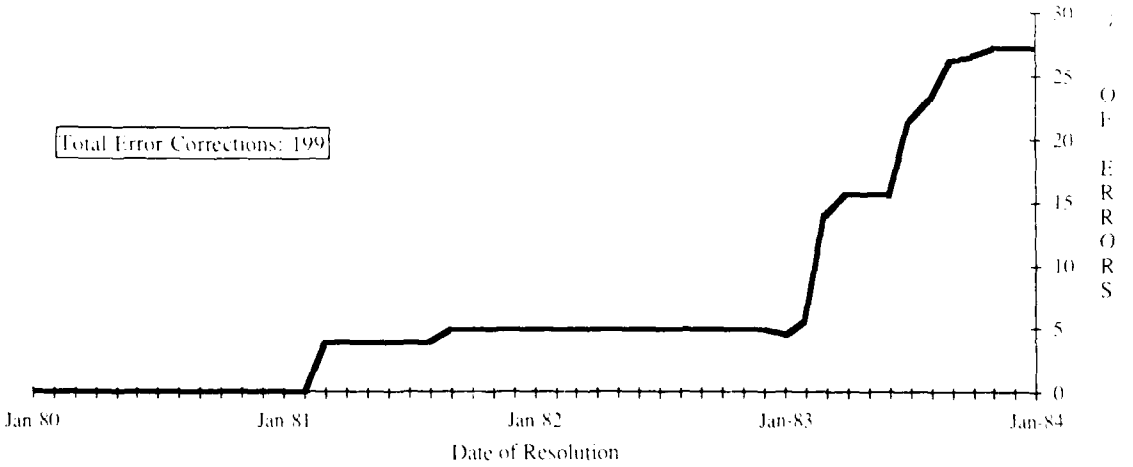

Fig. 9 — Correction or completion errors: percentage of error corrections

Ease of Change

A major objective of the SCR project is to produce a software design, code, and documentation set that can easily specify and implement changes. Figure 10 shows the distribution of effort required for understanding and incorporating the 297 accepted changes into the SCR project's design and documentation set; Fig. 11 shows the distribution for error corrections only. Only one of the 28 rejected CRFs was not implemented because the proposed change was considered to be not worth the effort. Most changes (81%) took an hour or less to understand and resolve; 98% took a day (i.e., 8 person hours) or less. Eighty-six percent of the error corrections took an hour or less to understand and

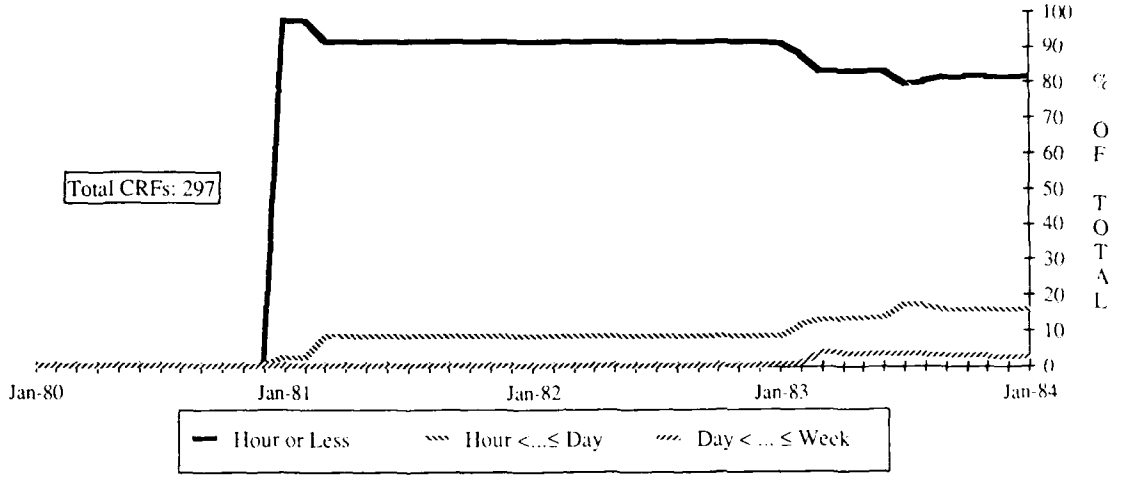

Fig. 10 — Accepted CRFs categorized by resolution effort

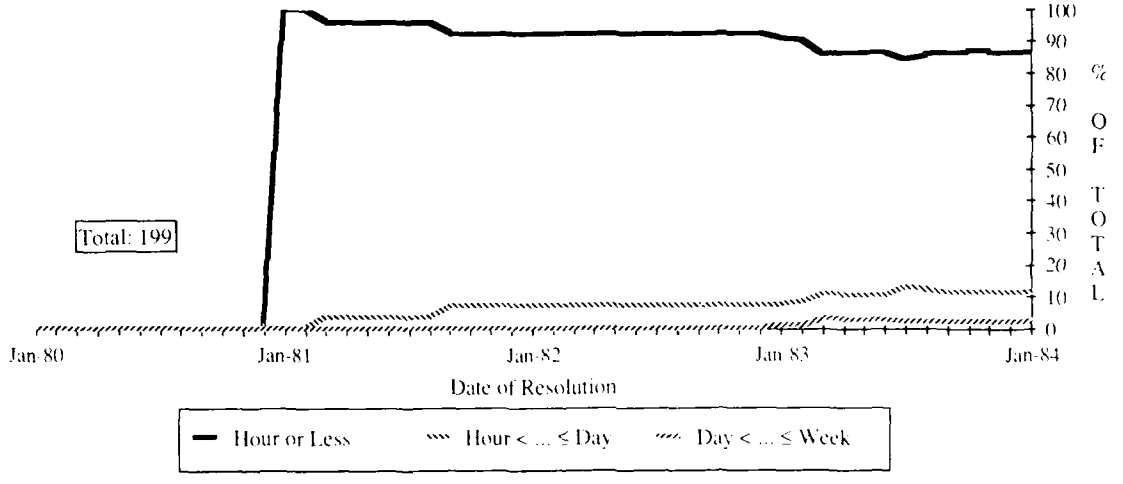

Fig. 11 — Error corrections categorized by resolution effort

resolve; 99% took a day or less. Although the data shown in Figs. 10 and 11 exhibit downward trends, these data suggest that SCR engineers are meeting their major objective of early changes and error corrections. For errors uncovered and corrected late in the life cycle of a NASA/Goddard Software Engineering Laboratory project, Basili and Perricone [15] report 36% of the error corrections took an hour or less; 55% took a day or less. For errors uncovered and corrected late in the WPADT project, Xu [22] reports 24% of the error corrections project took an hour or less and 80% took a day or less.

Figure 12 presents the cumulative average effort for all SCR changes and error corrections. There seems to be a stepwise growth in cumulative average change effort as the SCR project life cycle lengthens. This is consistent with Boehm's [27] data that show an exponential growth in cost to fix or change software for successive phases of the software life cycle. In terms of this result, the SCR project seems no different than other software development projects. Figure 13 presents the effort for an error correction based on number of days that the error is in the system. The figure "days in system" is the difference between CRF resolution date and the earliest issue date for the

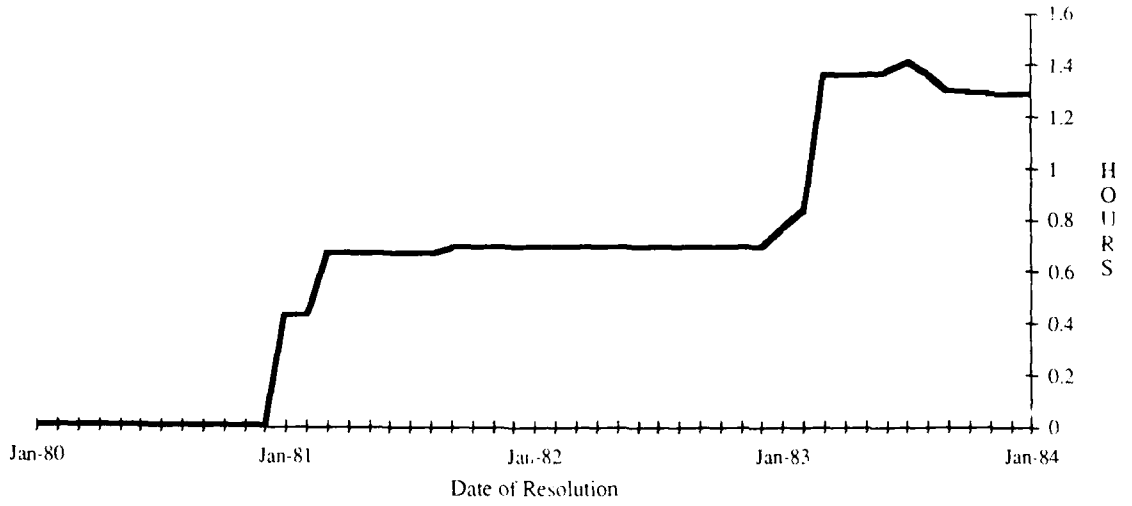

Fig. 12 — Cumulative average CRF effort

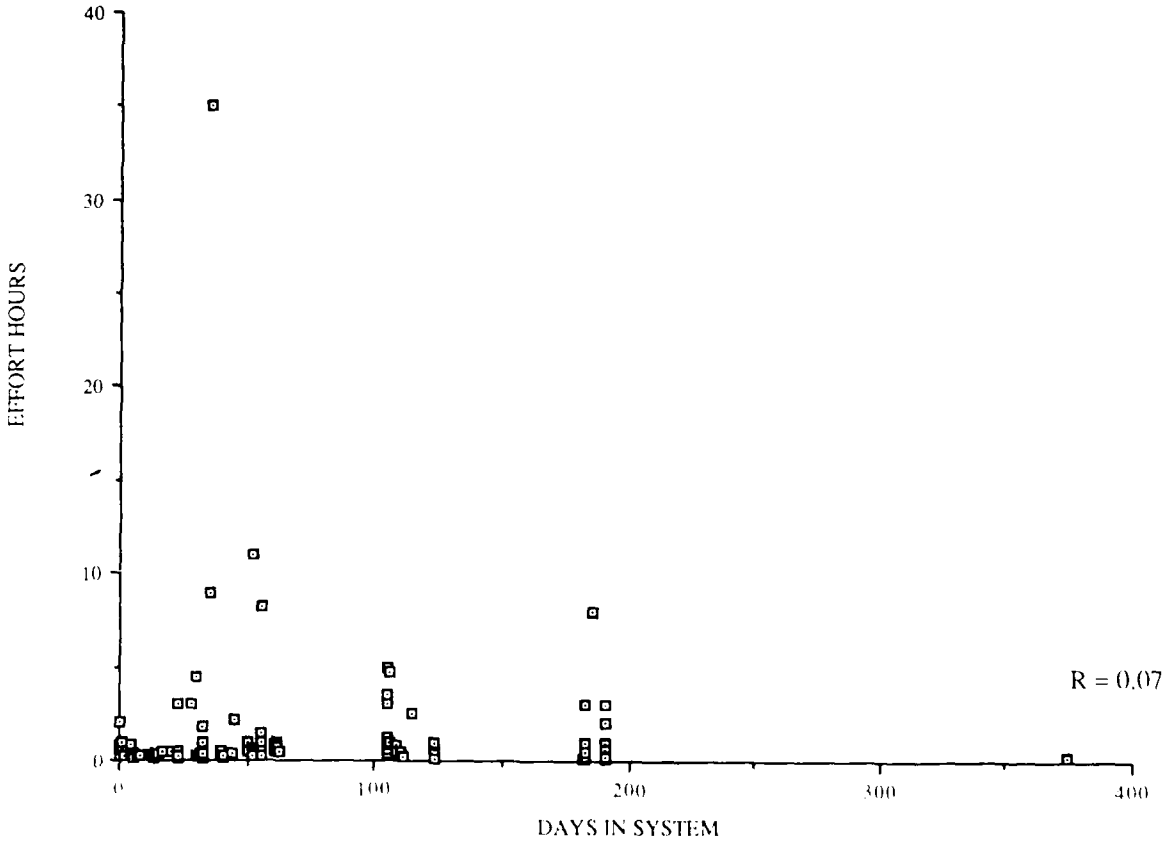

Fig. 13 — Duration of an error in the system

interface specifications containing the error. Boehm's data imply that the longer an error remains undetected and uncorrected in a system, the greater the cost of the eventual error correction. Surprisingly, this effect does not appear in the SCR data; the correlation between days in system and average effort is 0.07. There may be four reasons for this. The first is that SCR requirements change data are not included here. The second is that the changes reported here can be considered to be only design-phase changes, and more of the SCR projects life cycle might have to pass before any relationship appears. The third is that there are many very low effort changes. And the fourth is that the SCR methodology lessens the impact of long-term unresolved errors!

The information-hiding principle is used in the SCR project for identifying and specifying modules. A module is supposed to hide a likely changeable aspect of the A-7E flight software. This means that a module's interface specification must be written such that the hidden information is not revealed, that is, a module's hidden information is available only to the implementors of that module. The anticipated result is that when an expected change occurs only one module implementation (i.e., no interface) needs modification. Figure 14 presents the distribution for the number of modules updated by changes (i.e., the ripple effect of changes). A module is updated if its interface specification (implementation document, or code) is updated. A change is considered to update zero modules if updates are required in other documentation or in indexes and tables of contents associated with packaged sets of module specifications. Most changes (90%) updated zero or one modules, and this percentage is relatively constant. Figure 15 presents the proportion of changes that resulted in module interface updates. A module interface is updated if a change to its specification (or implementation document, or code) causes or would have conceivably caused a change to programs of other modules that use or would eventually use capabilities provided by the module. Examples of interface updates are the modification of a parameter type and the addition of a system-generated parameter. The percentage of changes that resulted in updated interface updates (56%) is growing. The percentage of changes updating two or more interfaces (12%) is also growing. These trends seem to suggest that a greater ripple effect and a more uniform distribution of change effort can be expected in the future.

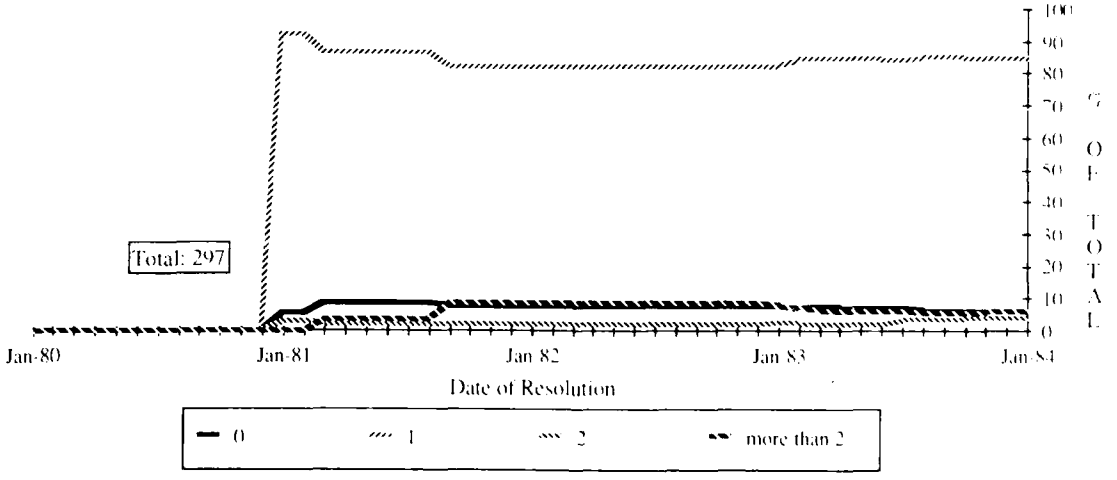

Fig. 14 — Accepted CRFs categorized by number of modules changed

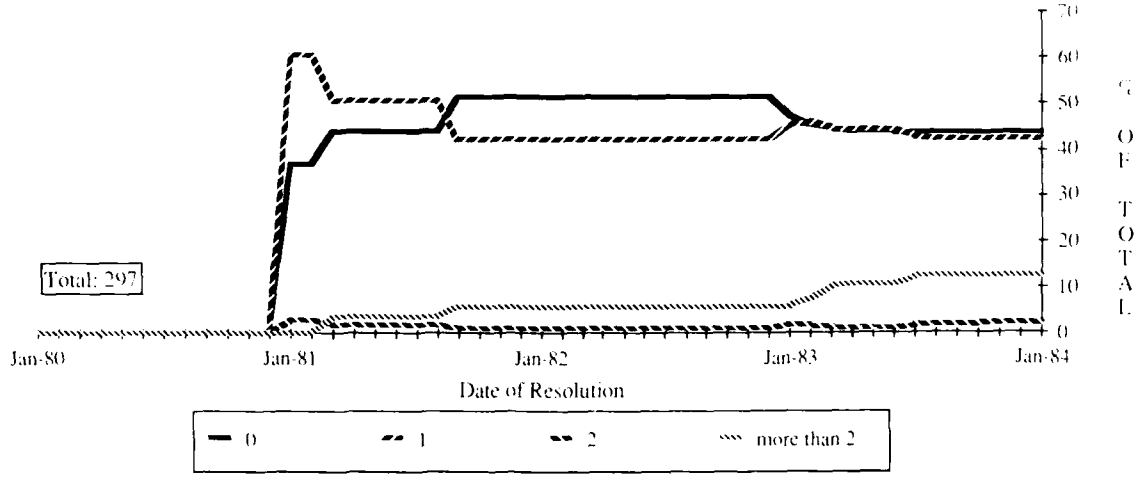

Fig. 15 — Accepted CRFs categorized by number of interfaces updated

Error Causes

Figure 16 shows the distribution of error causes. Thirty-three percent of the error corrections are clerical, that is, they are characterized as likely to have been made when the material was being typed. This percentage, which is growing, is large in comparison to other reported data, e.g., Basili and Perricone [15]. Weiss [24], however, has reported a 36% figure for an earlier NRL software project, the Architecture Research Facility.

The majority of errors (65%) have "other" causes. An examination of these causes shows that engineers attributed the errors to failings on their part. Thus, this percentage is close to the 68% figure for programmer error reported by Ostrand and Weyuker [25].

Only two errors (1%) were felt to be caused by poor SCR documentation! This contrasts to the 9% figure for poor documentation reported in Ref. 25. Either SCR engineers are reluctant to fault their documentation, or their documentation is quite good, or they simply tend to blame themselves for errors. The last 1% of errors had unknown causes.

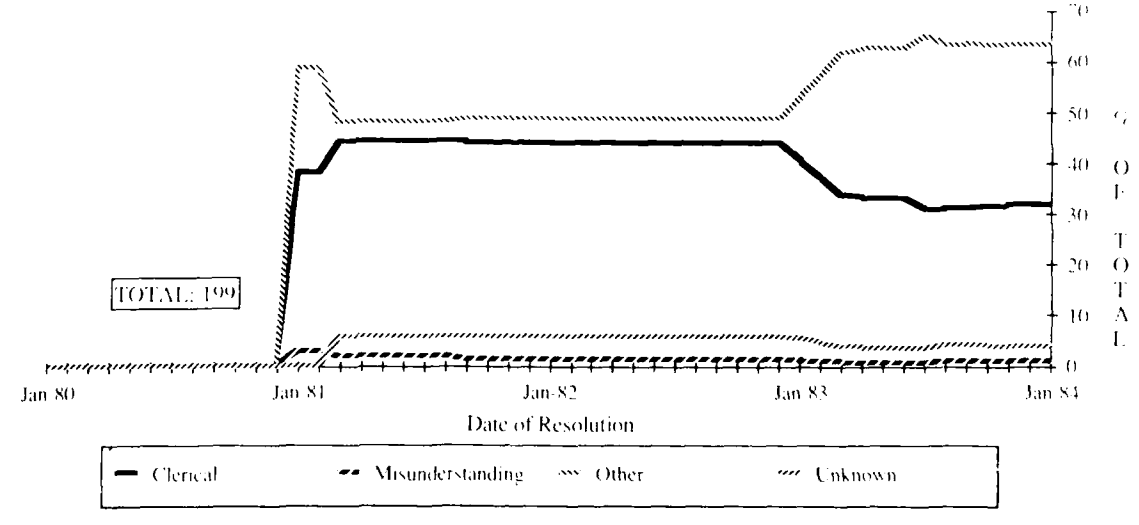

Fig. 16 — Error causes: percentage of total

Change Data Related to Personnel Activity Data

SCR project engineers report their activity weekly, using forms designed by software technology evaluation (STE) researchers. Figure 17 shows the ratio of the cumulative changes uncovered during a specific activity (i.e., design, code, and test) to the cumulative hours that were expended in that activity. Figure 18 shows the ratio of cumulative hours for changes uncovered during an activity to the cumulative activity hours. They show a similar pattern. Coding activity is the most "efficient" way to uncover needed modifications and errors, followed closely by testing activity. This is true only initially, however. In the long run, for the SCR project, design, code, and test activity are all equally efficient in terms of uncovering changes. However, the amount of coding (6504.25 hours) and testing (1487.5 hours) that accumulated by January 1984 is small compared to the amount of design (21741.75 hours).

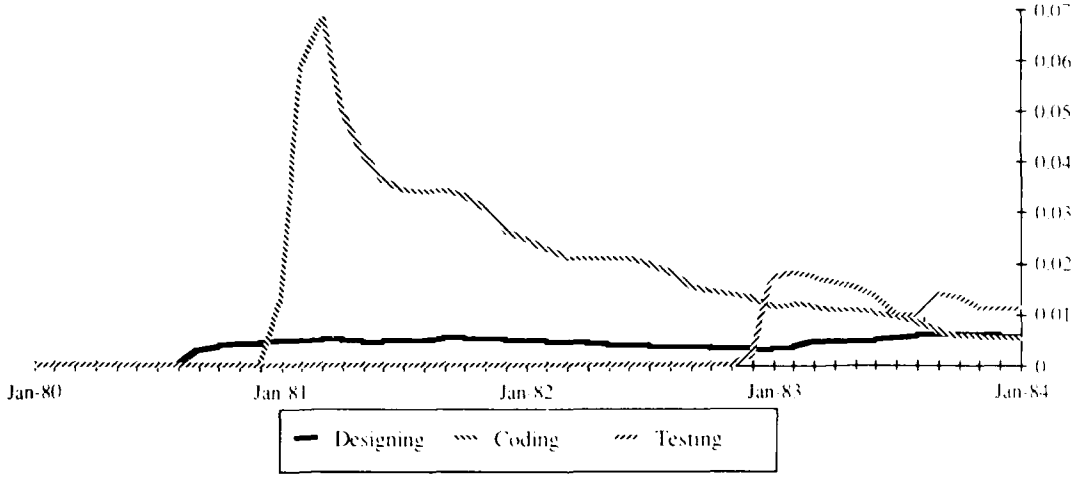

Fig. 17 — Ratio of cumulative CRFs over cumulative origination activity hours

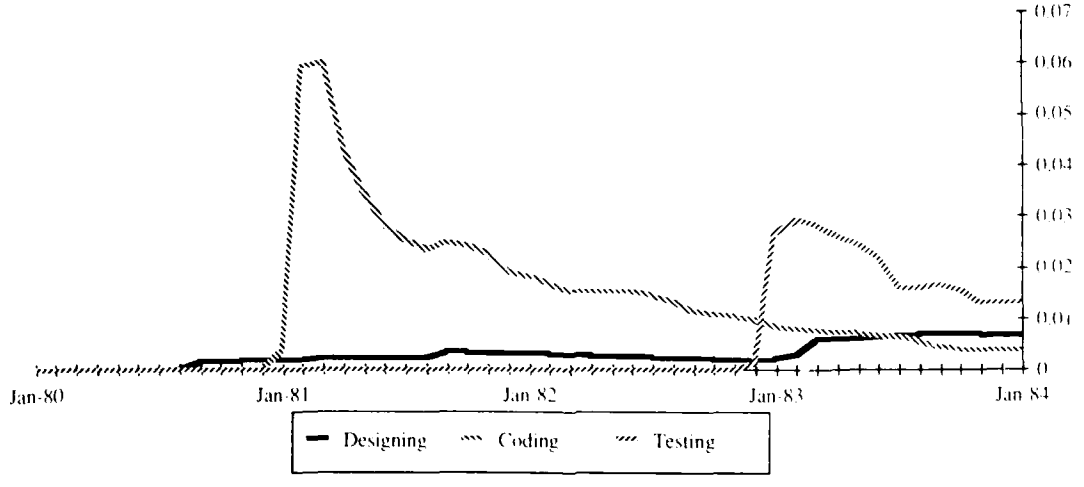

Fig. 18 — Ratio of cumulative CRF resolution effort by origination activity over cumulative activity hours

Figure 19 shows the proportion of error corrections for a project work month and the proportion of implemented changes for a work month (i.e., 160 person hours). Although they appear to be increasing, both ratios are small compared to the data reported by Weiss and Basili [17]. They report ~2 to 3 error corrections per work month and 4 to 8 changes per work month.

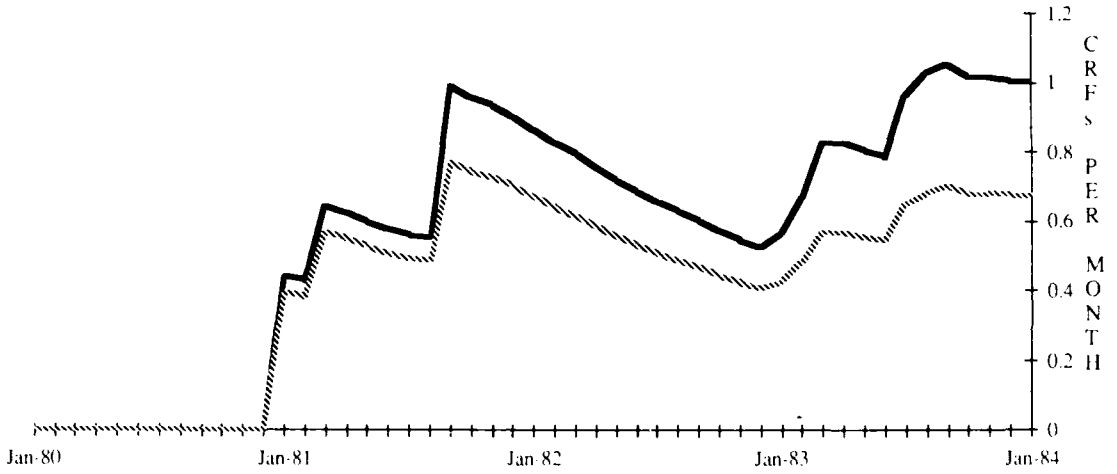

Fig. 19 — Ratio of cumulative error corrections and accepted CRFs to cumulative project months

DATA ANALYSES

A previous study of SCR project activity data [10] has defined the Progress Indicator Ratio (PIR). The PIR, which is a time-based ratio between a module's cumulative design discussing hours and cumulative design creating hours, consistently correlates with total design hours for the module. When the release dates for specification baselines are examined in conjunction with the PIR, the PIR seems to indicate incompleteness of baseline specifications. The appearance of a baseline before the PIR rises sharply or during a sharp rise seems to suggest that the baseline is probably far from complete. Module interface specifications seem to become reasonably stable only when the PIR becomes stable.

A major objection to the PIR is that it requires a data collection scheme that accurately captures intricate information about personnel activity during the design process. Even though this seems possible [26], few software development efforts can readily afford and tolerate the collection operation. Because many design efforts routinely record software change data, it would be desirable if information provided by the PIR could also be provided by change data. Figure 17 suggests a possible use of change data. Also, intuition suggests that a module's interface design would be unstable if people who were working on that design were generating many CRFs against the current version of the design or against the interface designs of other modules.

Table 3 lists some of the second-level modules of the multilevel hierarchy of information-hiding modules resulting from the SCR design activity [7]. These modules have interface specifications that have had one or more baselines, and each has been modified by one or more of the 325 CRFs. For each of the modules, time-based ratios between the number of CRFs resulting from module design activity and the cumulative module design hours can be computed and plotted. These are the Date of Origin PIR (DOOPIR) and the Date of Resolution PIR (DORPIR), based on CRF date of origin and resolution, respectively. Table 4 is a summary of the data underlying these ratios; specifically, they

Table 3—Abbreviations and Names of Second-Level Software Modules

Abbreviation	Name
AT	Applications Data Type
DI	Device Interface
EC	Extended Computer
FD	Function Driver
SS	Shared Services

Table 4—Total CRFs and Design Hours Through December 1983

Module	CRFs Resulting from Design	Earliest CRF Date of Origin	Design Hours
AT	2	Mar 81	1083.75
DI	11	Sept 80	2859.00
EC	119	Mar 81	7477.50
FD	27	Sept 80	1235.05
SS	6	Jan 81	1848.45

are the number of CRFs that resulted during design work on the module, the date of origin of the earliest of these CRFs, and the total design hours for each module.

Date of Origin PIR

For each module, the DOOPIR is defined as the ratio between the cumulative CRFs uncovered during design of the module by date of origin and cumulative design hours for the module. Figures 20 to 24 show DOOPIRs for each module. The vertical lines in these figures indicate issue dates for module specification baselines. Table 5 shows Pearson product moment correlation coefficients r and coefficients of determination r^2 between DOOPIRs and the original PIRs for each module [27]. The time period over which correlations are computed begins with the date of origin of the earliest CRF, as presented in Table 4.

Date of Resolution PIR

The DORPIR is the same DOOPIR except that CRF date of resolution is used rather than date of origin. Figures 25 through 29 show DORPIRs for each module. Again, vertical lines indicate baseline issue dates. Table 6 shows Pearson product moment correlation coefficients r and coefficients of determination r^2 between DORPIRs and the original PIRs for each module [27]. The time period over which correlations are computed is the same as for the DOOPIR. Even though the date of resolution occurred after the date of origin, hours of resolution effort include origination time plus subsequent change time.

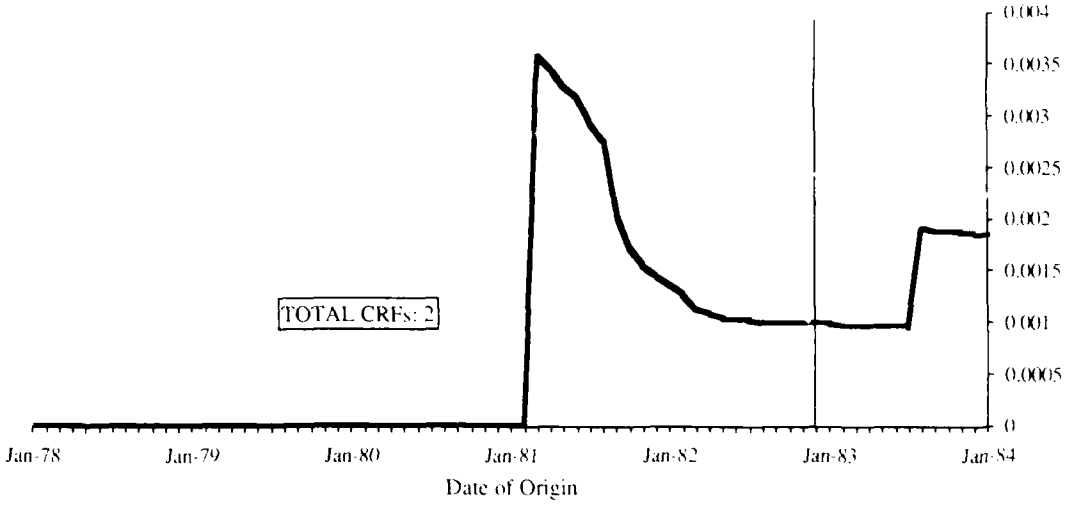

Fig. 20 — Date of origin PIR for AT

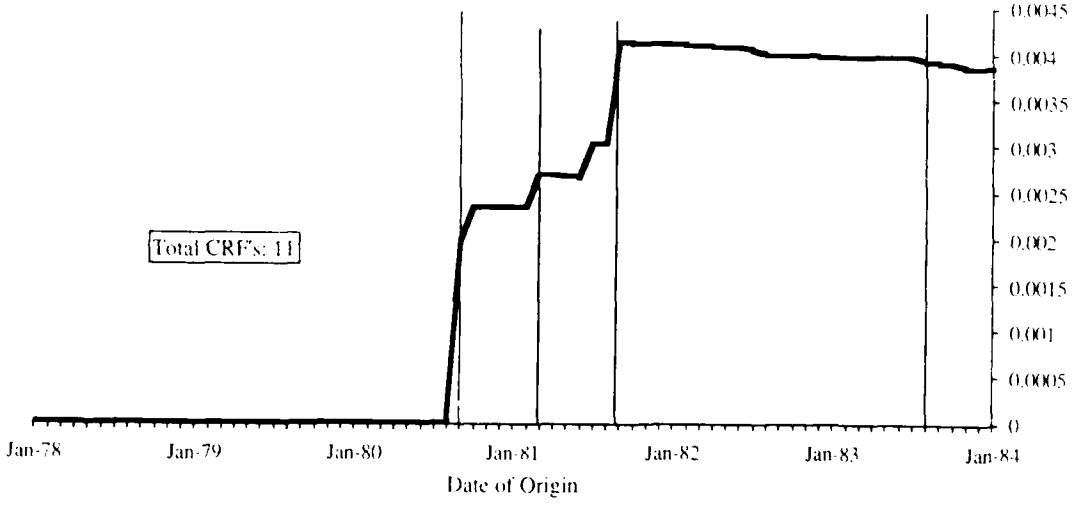

Fig. 21 — Date of origin PIR for DI

Table 5—Pearson Correlation Coefficients Between DOOPIR and PIR

Module	r	r^2
AT	−0.610*	0.372*
DI	0.727	0.528
EC	0.985	0.970
FD	−0.679	0.461
SS	−0.478*	0.228*

*Not significant at the $p \geq .05$ level.

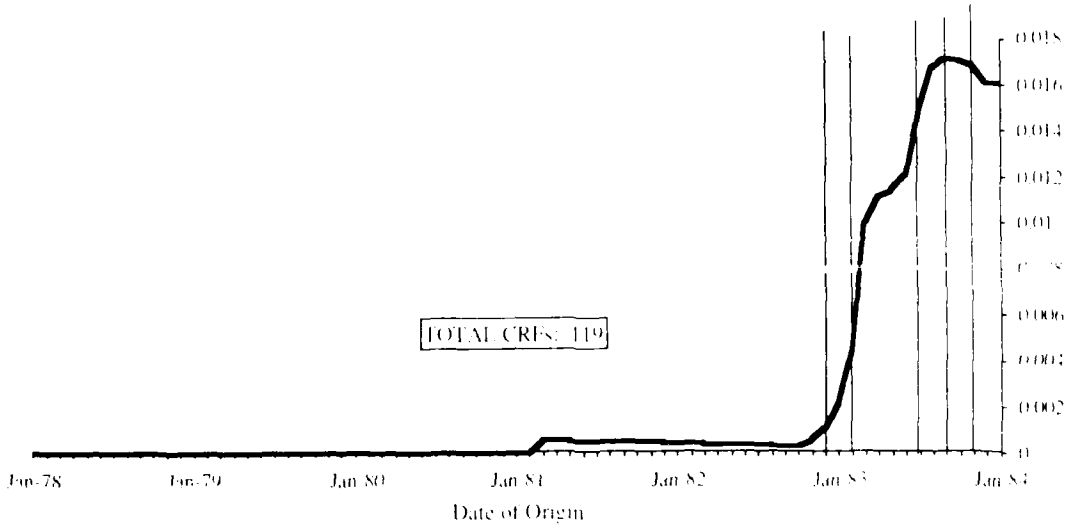

Fig. 22 — Date of origin PIR for EC

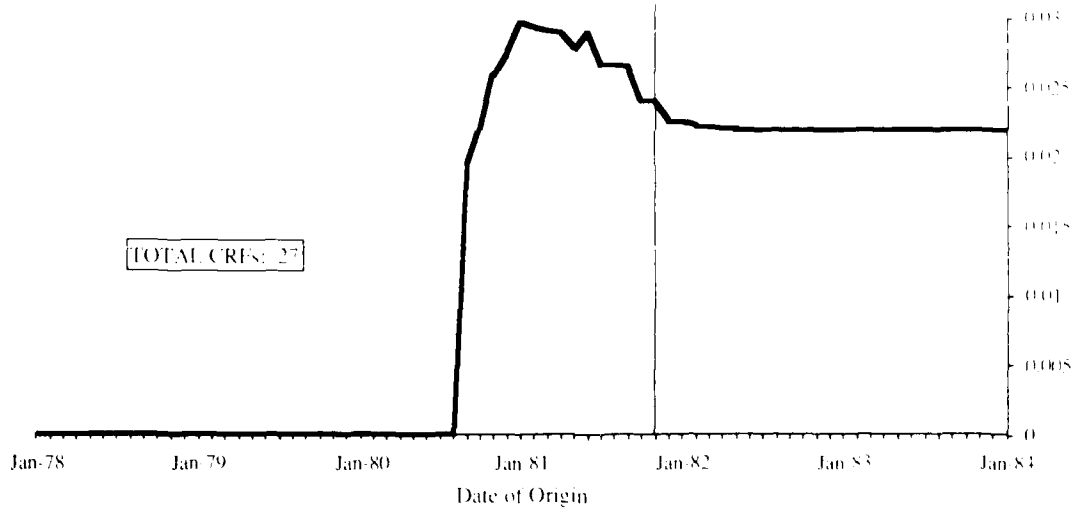

Fig. 23 — Date of origin PIR for FD

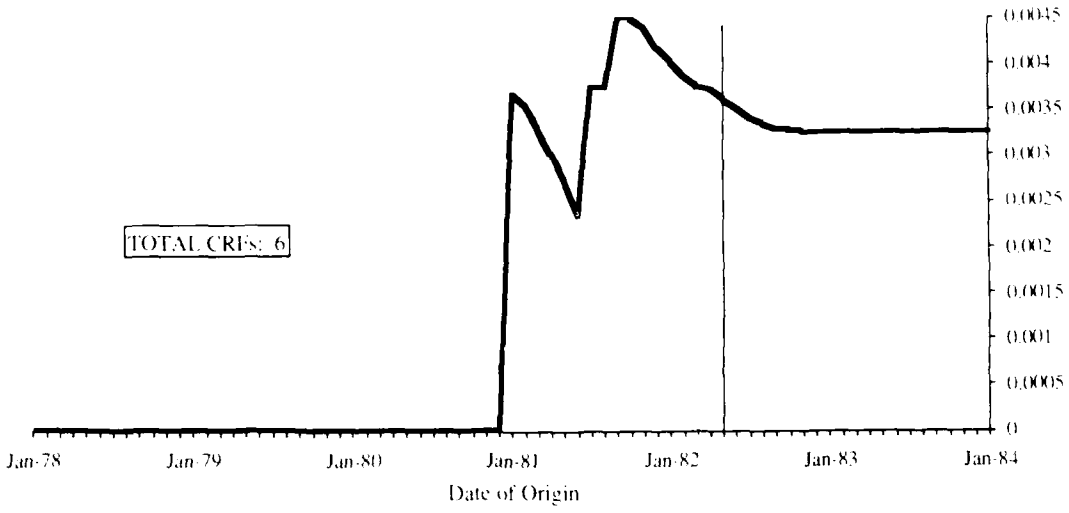

Fig. 24 — Date of origin PIR for SS

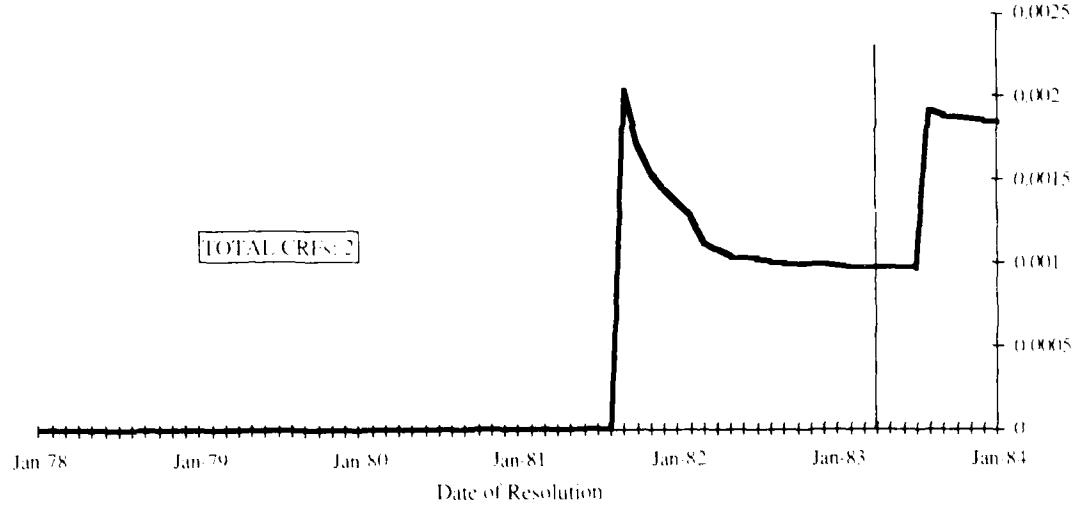

Fig. 25 — Date of resolution PIR for AT

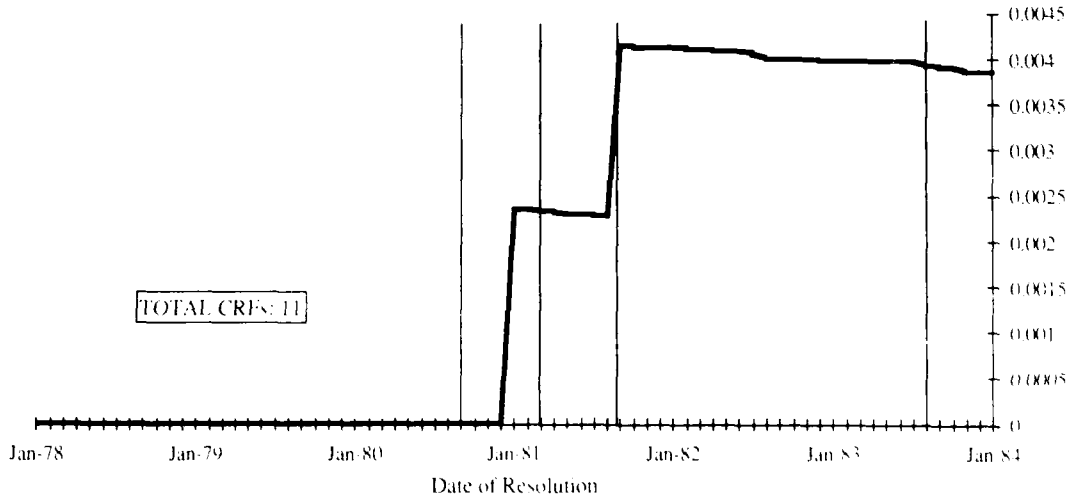

Fig. 26 — Date of resolution PIR for DI

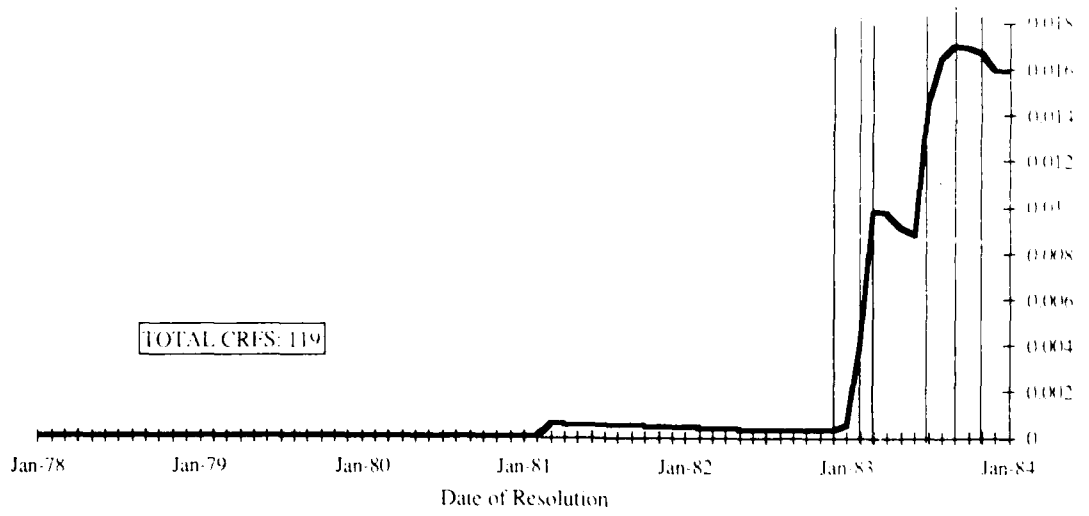

Fig. 27 — Date of resolution PIR for EC

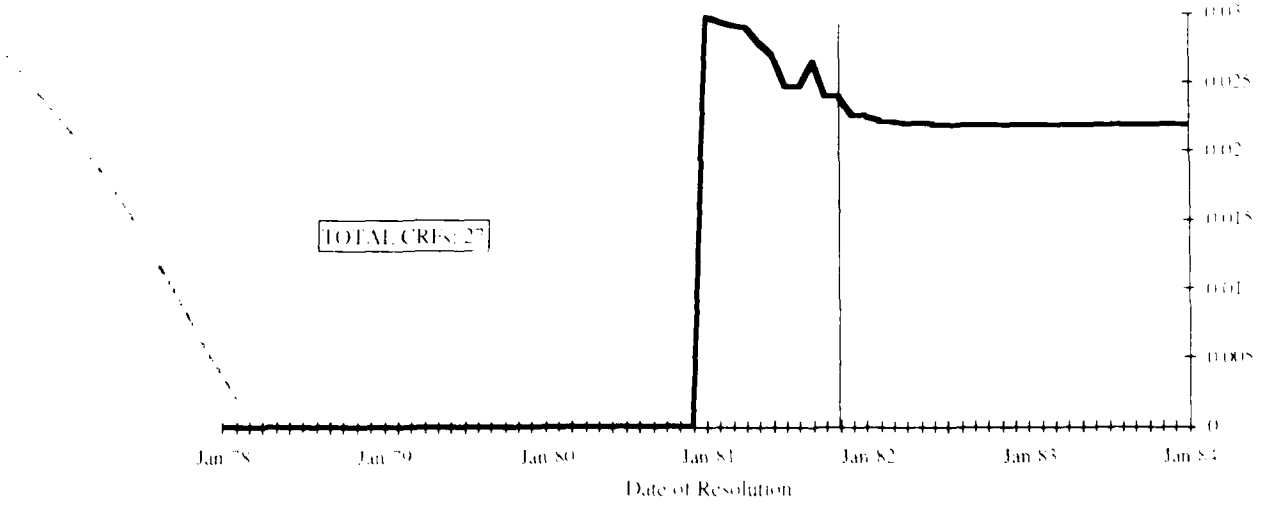

Fig. 28 — Date of resolution PIR for FD

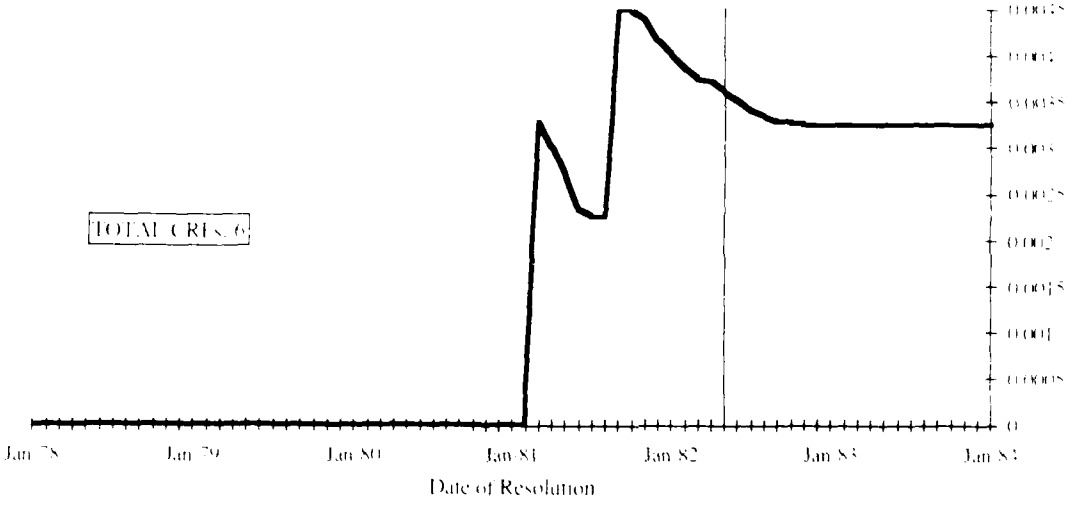

Fig. 29 — Date of resolution PIR for SS

Table 6—Pearson Correlation Coefficients Between DORPIR and PIR

Module	r	r^2
AT	0.391*	0.152*
DI	0.698	0.487
EC	0.971	0.943
FD	0.709	0.503
SS	−0.472*	0.223*

*Not significant at the $p \geq .05$ level.

RESULTS AND CONCLUSIONS

An overview of the SCR project's early change data with respect to customary concerns and a time-based view shows these major patterns:

- There is a high proportion of error corrections and error correction effort, although time-based plots of these statistics show that both are decreasing.
- The percentage of error corrections that involve completing or correcting a prior change is far higher than has ever been reported, and this percentage is increasing.
- The percentage of changes that took a day or less to resolve is extremely large, but this is decreasing. Consistent with this decrease is a stepwise growth in average change effort, a growth in the percentage of changes that involve modifying module interfaces, and a growth in the percentage of changes involving two or more module interfaces.
- Surprisingly, no relationship has been shown between change effort and number of days that an error exists in the documentation.
- Very few errors have been attributed to poor project documentation.
- Coding activity, followed by testing activity, is the most efficient way of uncovering needed modifications and error corrections. In the long run, however, design, code, and test activity appear to be equally efficient.

Analyses of the design CRF data suggest that, in some cases, fairly simple change and personnel activity data can be used as an alternative to the originally proposed PIR. The DOOPIRs and the DORPIRs for modules with a significant number of design changes show a strong relationship to the original PIRs. Ten CRFs can be considered a reasonable threshold for sensitivity. The DOOPIR explains 52%, 97%, and 46% of the variation in the original PIRs for the DI, EC, and FD modules, the DORPIR explains 49%, 94%, and 50% of the variations for these same modules.

When issue dates for published baselines are superimposed on the DOOPIR and DORPIR plots, patterns similar to if not even more sensitive than those observed with the original PIR are observed. For module designs that have been specified with only one or two baselines, a prior instability with the DOOPIR and DORPIR, a downward trend, issuance of the baseline, and then relative stability are seen. For other modules, this pattern is lacking one or more of the earlier baselines. In other words, both the DOOPIR and the DORPIR appear to indicate incompleteness in the interface specifications. If these ratios have not surged and then turned downwards prior to appearance of a baseline and subsequently stabilized, the design of the module's interface probably is not complete, despite personnel claims and published documents.

There are two drawbacks to the DOOPIR and DORPIR. They are later indicators of design progress than the original PIR, and they are based heavily on the responsiveness and timeliness of a project's change control process. If changes are not resolved promptly, the relationships between these ratios and design progress are weakened.

Finally, we do not claim that the DOOPIR or the DORPIR are measures of the completeness of an interface design. There may be many reasons why the ratios stabilize (e.g., personnel have been assigned to another module) or have taken vacations. However, the ratios do seem to indicate when work on an interface is not complete. If completion is claimed prior to a downward trend and subsequent stability, more work probably must be done.

ACKNOWLEDGMENTS

We are especially indebted to Paul Clements, the lead SCR software engineer, who patiently assisted CRF validators in resolving problems that were encountered. Also, to Ms. Kathryn Kragh who for several years entered change and activity data into a computer database, checked the accuracy of each entry, and updated everything when, for example, the names of modules changed. Without her diligence, this report could not have been written.

REFERENCES

1. P.C. Clements, "Software Cost Reduction Through Disciplined Design," *1984 Review*, Naval Research Laboratory, pp. 79-87 (1985).

2. B. Meyer, "On Formalism in Specifications," *IEEE Software* **2**(1), 6-27 (1985).

3. D.L. Parnas "On the Criteria to Be Used in Decomposing Systems into Modules," *Commun. ACM* **15**, 1053-1058 (1972).

4. D.L. Parnas, "Use of Abstract Interfaces in the Development of Software for Embedded Systems," NRL Report 8047, June 1977.

5. E.W. Dijkstra, "Cooperating Sequential Processes," in *Programming Languages*, F. Genuys, ed. (Academic Press, New York, 1968), pp. 43-112.

6. K.L. Heninger, J.W. Kallander, J.E. Shore, D.L. Parnas, and Staff, "Software Requirements for the A-7E Aircraft," NRL Memorandum Report 3876, Nov. 1978.

7. K.H. Britton and D.L. Parnas "A-7E Module Guide," NRL Memorandum Report 4702, Dec. 1981.

8. A. Parker, K.L. Heninger, D. Parnas, and J. Shore, "Abstract Interface Specifications for the A7-E Device Interface Module," NRL Memorandum Report 4385, Nov. 1980.

9. P.C. Clements, R.A. Parker, D.L. Parnas, and J. Shore, "A Standard Organization for Specifying Abstract Interfaces" NRL Report 8815, June 1984.

10. A.F. Norcio and L.J. Chmura, "Design Activity in Developing Modules for Complex Software," in *Empirical Studies of Programmers*, E. Soloway and S. Iyengar, eds. (Ablex Publishing Corporation, Norwood, New Jersey, 1986).

11. L.J. Chmura and D.M. Weiss, "The A-7E Software Requirements Document: Three Years of Change Data," NRL Memorandum Report 4938, Nov. 1982.

12. V.R. Basili and D.M. Weiss "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Eng.* **SE-10**(6), 728-738 (1984).

13. L.J. Chmura, "Proposed New Design and Code Change Report Form (CRF) for the Software Cost Reduction (SCR) Project," NRL Technical Memorandum 7590-34:LC (1983).

14. E.B., Swanson, "The Dimensions of Maintenance," *Proceedings of the Second International Conference on Software Engineering*, IEEE Computer Society, (1976), pp. 492-497.

15. V.R. Basili and B.T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Commun. ACM* **27**, 42-52 (1984).

16. A. Endres, "An Analysis of Errors and Their Causes in System Programs," *IEEE Trans. Software Eng.* **SE-1**(2), 140-149 (1975).

17. D.M. Weiss and V.R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," *IEEE Trans. Software Eng.* **SE-11**(2), 157-168 (1985).

18. R. Day, "A History of Software Maintenance for a Complex U.S. Army Battlefield Automated System," (1985), pp. 181-187.

19. M.L. Shooman and M.I. Bolsky, "Types, Distribution, and Test and Correction Times for Programming Errors," Proceedings of the International Conference on Reliable Software, ACM SIGPLAN Notices (1975), pp. 347-357

20. B.R. Lientz, E.B. Swanson and G.E. Tompkins, "Characteristics of Application Software Maintenance," *Commun. ACM* **21**(6), 466-471 (1978).

21. D.M. Weiss, "A Comparison of Errors in Different Software-Development Environments," NRL Report 8598, July 1982.

22. R.Z. Xu, "An Empirical Investigation: Software Errors and Their Influence Upon Development of WPADT," *Proceedings of COMPSAC 85*, IEEE Computer Society, pp. 4-8 (1985).

23. B.W. Boehm, *Software Engineering Economics* (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981).

24. D.M. Weiss, "Evaluating Software Development by Error Analysis: The Data from Architecture Research Facility," *J. Systems Software* **1**(1), 57-70 (1979).

25. T.J. Ostrand and E.J. Weyuker, "Software Error Data Collection and Categorization," Engineering Workshop (1982).

26. L.J. Chmura and A.F. Norcio, "Accuracy of Software Activity Data: The Software Cost Reduction Project," NRL Report 8780, Dec. 1983.

27. W.J. Dixon and F.J. Massey, Jr., *Introduction to Statistical Analysis* (McGraw-Hill Book Co., Inc., New York, 1969).